ResEntSG: restoration entropy estimation for dynamical systems via Riemannian metric optimization

Article (Published Version)

SoftwareX 15 (2021) 100743

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

ResEntSG: Restoration entropy estimation for dynamical systems via Riemannian metric optimization

Christoph Kawan [a], Sigurdur Freyr Hafstein [b,*], Peter Giesl [c]

[a] Institute of Informatics, LMU Munich, Oettingenstraße 67, 80538 München, Germany
[b] Science Institute, University of Iceland, Dunhagi 3, 107 Reykjavík, Iceland
[c] Department of Mathematics, University of Sussex, Falmer, BN1 9QH, United Kingdom

ARTICLE INFO

Article history:
Received 29 April 2021
Received in revised form 4 June 2021
Accepted 9 June 2021

Keywords:
Restoration entropy
Subgradient algorithm
Geodesic convexity
Riemannian metric
Dynamical systems
C++
Scientific computing

ABSTRACT

In the remote state estimation problem, an observer reconstructs the state of a dynamical system at a remote location, where no direct sensor measurements are available. The estimator only has access to information sent through a digital channel. The notion of restoration entropy provides a way to determine the smallest channel capacity above which an observer can be designed that observes the system without a degradation of the initial estimation error. In general, restoration entropy is hard to compute. We present a class library in C++, that estimates the restoration entropy of a given system by computing an adapted metric for the system. The library is simple to use and implements a version of the subgradient algorithm for geodesically convex functions to compute an optimal metric in a class of conformal metrics. Included in the software are three example systems to demonstrate the use and efficacy of the library.

Code metadata

Current code version	v1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00086
Legal Code License	GNU General Public License v3.0
Code versioning system used	none
Software code languages, tools, and services used	C++, Armadillo
Compilation requirements, operating environments & dependencies	Unix-like systems
Link to developer documentation/manual	
Support email for questions	shafstein@hi.is

1. Motivation and significance

Networked control systems are often composed of a large number of spatially distributed subsystems which share a common wireless communication network for information transfer. Prominent applications include: cooperative driving of connected cars, the coordination of a swarm of drones, the control of unmanned surveillance and rescue submarines, and robots playing football. Designing communication and control protocols for such applications is challenging, because they violate standard assumptions in classical control theory due to several imperfections, one of which is a limitation of the available data rate. In all practical applications, controllers first compute an estimate of the current state of the system before they determine a control action based on this estimate. These facts motivate the problem of the design of observers which receive sensory data over rate-limited channels. In particular, it is of interest to find the data-rate limit under which such observers can be designed.

The paper [1] introduced the notion of *restoration entropy (RE)*, a quantity characterizing the data-rate limit for so-called *regular* or *fine observability* of nonlinear systems over digital channels. Here, the estimation error at any time is bounded above by a fixed multiple of the initial error or additionally converges to zero exponentially (in the case of fine observability).

* Corresponding author.
E-mail address: shafstein@hi.is (Sigurdur Freyr Hafstein).

https://doi.org/10.1016/j.softx.2021.100743

The RE is defined for systems given by

$$\underbrace{x(t+1)=f(x(t))}_{\text{discrete time}} \quad \text{or} \quad \underbrace{\dot{x}=f(x)}_{\text{continuous time}}, \quad x\in\mathbb{R}^n$$

where we only consider initial states from a compact, forward-invariant set $K \subset \mathbb{R}^n$. In [2], we have developed a numerical algorithm for the estimation of the RE. This algorithm is based on a characterization of RE expressed in terms of the singular values of the time-one map (for discrete-time systems) or their infinitesimal counterparts (for continuous-time systems), computed with respect to a Riemannian metric on K. Here, one has to consider the infimum of a certain singular value function, taken over all such metrics. Hence, the computation of RE can be regarded as an infinite-dimensional optimization problem on the space of all Riemannian metrics on K.

The algorithm developed in [2] solves a constrained version of this optimization problem, where we restrict the domain to the class of conformal metrics $P(x) = \mathrm{e}^{r(x)}p$ with $r(x)$ a polynomial of bounded degree and p a positive definite symmetric $n \times n$ matrix. This restricted problem can be solved via a subgradient algorithm due to the observation that the function to be minimized is geodesically convex with respect to $(a, p) \in \mathbb{R}^{|I|} \times \mathcal{S}_n^+$, where a is the coefficient vector of the polynomial, I an index set, and $r(x) = r_a(x)$. Hence, we have to deal with a geodesically convex problem on the product space $\mathbb{R}^{|I|} \times \mathcal{S}_n^+$, which is a complete Riemannian manifold with nonpositive sectional curvature, when $\mathcal{S}_n^+$ is equipped with the so-called trace metric [3]. The classical subgradient algorithm has been extended to geodesically convex problems on Riemannian manifolds in [4,5], and corresponding convergence results have been proven.

In this paper, we present and explain the code for the subgradient algorithm developed in [2], which we have successfully applied to three nonlinear chaotic systems (Hénon, harmonically forced bouncing ball and Lorenz).

2. Algorithms

Due to space constraints, we only give a minimalistic description of the algorithms used; for a detailed discussion, see [2]. We consider a dynamical system,

$$\underbrace{x(t+1)=f(x(t))}_{\text{discrete time}} \quad \text{or} \quad \underbrace{\dot{x}=f(x)}_{\text{continuous time}}, \quad x\in\mathbb{R}^n \tag{1}$$

on a compact, forward-invariant set $K \subset \mathbb{R}^n$ that is the closure of its interior. We assume that f is C^1, set $A(x) := \mathrm{D}f(x)$, and in the discrete-time case we additionally assume that $A(x)$ is invertible for all $x \in K$. We search for a Riemannian metric M on K of the form

$$M(x) = \mathrm{e}^{r_a(x)}p, \quad p \in \mathcal{S}_n^+, \quad r_a(x) = \sum_{\alpha\in I} a_\alpha x^\alpha, \tag{2}$$

that gives an optimal upper bound on the restoration entropy of the system (1). Here, $\mathcal{S}_n^+$ denotes the set of positive definite $n \times n$ symmetric matrices (we use the notation $\mathcal{S}_n$ for the space of all symmetric $n \times n$ matrices), $I \subset \mathbb{N}_0^n$ is a finite set of multi-indices, $a_\alpha \in \mathbb{R}$, and $x^\alpha := x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. This is achieved by minimizing the function $(a, p) \mapsto \max_{x\in K} g(x; a, p)$, defined in (3) (for discrete time) and (4) (for continuous time) below.

The search for the optimal metric is performed using the Riemannian subgradient algorithm on the Riemannian product manifold $\mathbb{R}^{|I|} \times \mathcal{S}_n^+$, where $\mathbb{R}^{|I|}$ is equipped with the usual Euclidean metric and $\mathcal{S}_n^+$ is equipped with the *trace metric* (cf. e.g. [3, 6])

$$\langle v, w\rangle_p := \mathrm{tr}(p^{-1}vp^{-1}w) \quad \text{for all } p \in \mathcal{S}_n^+,\ v, w \in T_p\mathcal{S}_n^+ = \mathcal{S}_n.$$

For this, we use the subgradient method for geodesically convex functions on $\mathbb{R}^{|I|} \times \mathcal{S}_n^+$, cf. [4,5]. The essence of the method is as follows:

1. Fix the number of iterations N, set $j = 0$ and start with some initial value $(a_j, p_j) = (a_0, p_0) \in \mathbb{R}^{|I|} \times \mathcal{S}_n^+$, typically $a_0 = 0$ and $p = I$; the $(n \times n)$ identity matrix.

2a. For a discrete-time system, find an $x^* \in K$ that maximizes

$$g(x; a_j, p_j) := \max_{0\le k\le n} \sum_{i=1}^{k} \log \alpha_i\Big(\mathrm{e}^{\frac{1}{2}[r_{a_j}(f(x))-r_{a_j}(x)]}p_j^{\frac{1}{2}}A(x)p_j^{-\frac{1}{2}}\Big), \tag{3}$$

where $\alpha_1(B) \ge \alpha_2(B) \ge \cdots \ge \alpha_n(B)$ denote the singular values of the matrix B in descending order and $\sum_{i=1}^{0} \ldots = 0$. Set $b_j = g(x^*; a_j, p_j)$.

2b. For a continuous-time system, find an $x^* \in K$ that maximizes

$$g(x; a_j, p_j) := \max_{0\le k\le n} \sum_{i=1}^{k} \zeta_i^{\mathrm{e}^{r_{a_j}(\cdot)}p_j}(x), \tag{4}$$

where $\zeta_1^B(x) \ge \zeta_2^B(x) \ge \cdots \ge \zeta_n^B(x)$ are the solutions of the algebraic equation

$$\det[B(x)A(x) + A(x)^\top B(x) + \dot{B}(x) - \lambda(x)B(x)] = 0 \tag{5}$$

and $\dot{B}(x)$ denotes the derivative of B along the solutions of $\dot{x} = f(x)$. Set $b_j = g(x^*; a_j, p_j)$.

3. Compute a subgradient $s = (s_1, s_2) \in T_{a_j}\mathbb{R}^{|I|} \times T_{p_j}\mathcal{S}_n^+ = \mathbb{R}^{|I|} \times \mathcal{S}_n$ of the mapping $(a, p) \mapsto g(x^*; a, p)$ at (a_j, p_j), see [2, (T3) in Sec. 5.1, 5.2] for the formulas.

4. For a time step $t_{j+1} > 0$, update the metric, or equivalently the position in $\mathbb{R}^{|I|} \times \mathcal{S}_n^+$, by $(a_{j+1}, p_{j+1}) = (a_j, p_j) - t_{j+1}n(s)$, where $n(s)$ is a normalized subgradient, increase j by one, and go back to Step 2 if $j < N$.

Each b_j in the algorithm is an upper bound on the RE and the best bound obtained is thus $\min_{j\le N} b_j$. Note that in general the sequence (b_j) is not monotonically decreasing. However, $\min_{j\le N} b_j$ converges to the optimal value as $N \to \infty$ if the time steps t_j are chosen such that $\sum_{j=1}^{\infty} t_j = \infty$ and $\sum_{j=1}^{\infty} t_j^2 < \infty$; a generic choice is $t_j = \alpha/(\beta + j)$ for some constants $\alpha > 0$ and $\beta \ge 0$.

Usually, the normalized subgradient $n(s)$ is simply taken to be $s/|s|$, but it is easy to see that any normalization $n(s)$ fulfilling $n_1 \le |n(s)| \le n_2$ for some constants $n_1, n_2 > 0$ independent of $s \ne 0$ will do the job. This is important because in our experiments we often find subgradients $s = (s_1, s_2)$ where $|s_1|$ and $|s_2|$ differ in several orders of magnitude and we obtain much faster convergence with $n(s) = (s_1/|s_1|, s_2/|s_2|)$. Note that $|(s_1/|s_1|, s_2/|s_2|)| = \sqrt{2}$.

3. Software description

ResEntSG is written in C++ and requires the *Armadillo C++ library* for linear algebra and scientific computing [7,8]. The example program *ResEntSGex* is typically run from the command line on a unix-based operating system, but we have verified that it also runs on Windows machines provided that the appropriate libraries are installed and linked. The execution of the example program to analyze a particular dynamical system generates two different files; e.g. for the Lorenz system:

- *LorenzEnt.txt* lists the upper bounds b_j of the restoration entropy from the initial values for p and r_a (0th iteration) to the last iteration. This file can be used to plot the estimate of the restoration entropy as a function of the iteration.
- *LorenzRes.txt* contains a summary of the results of the optimization. Example:

```
4000 iterations computed in 948 sec. : t_j =
    2/(j+0)
Best estimate of Restoration Entropy
    17.06379797426958
obtained in iteration 3718 with
a : (pow of x_1, pow of x_2, etc. )  :
    coefficient
( 1 0 0 ) : -0.0002372629969014084
( 0 1 0 ) : 0.0001951047281115719
( 0 0 1 ) : -0.2164241897811799
( 2 0 0 ) : 0.005518972770879637
( 1 1 0 ) : -0.003216928342620638
( 1 0 1 ) : 0.0001017761736674807
( 0 2 0 ) : 0.003663514414820466
( 0 1 1 ) : -0.0001621269969033825
( 0 0 2 ) : 0.003153319298490052

p :
2.100294257567501  -0.05928031113919301
    -0.007544500744831841
-0.05928031113919305  0.7310528027161857
    0.003440974905667806
-0.007544500744831843  0.003440974905667789
    0.6528204625830214
```

3.1. Software architecture

The source code is divided into the following separate files and classes:

- *EntSys.h* and *EntSys.cpp* contain the definitions and the implementations of methods for the virtual base class `EntSys` and its derived classes `EntSysCONT` (continuous time) and `EntSysDISC` (discrete time). The class `EntSys` contains the variables and methods that are common to continuous- and discrete-time systems. Its purely virtual methods are different for the continuous- and discrete-time case and are implemented in the corresponding derived classes.
- *DynSys4EntSys.h* and *DynSys4EntSys.cpp* contain the definition of the prototype class `DynSys4EntSys`, which should be inherited by all concrete dynamical systems. Further, it contains the three examples of concrete systems from [2].
- *Maximization.h* and *Maximization.cpp* contain the code used for the maximization of the function $x \mapsto g(x; a, p)$ from (3) or (4) and the definition and implementation of the class `xGrid`.
- *main.cpp*: the entry point for the three examples from [2] demonstrating the use of the class library.

3.2. Software functionalities

Our software computes an optimal upper bound for the restoration entropy on a forward-invariant set of a dynamical system, that can be obtained by a generic Riemannian metric of the form $M(x) = e^{r_a(x)}p$, where $p \in \mathcal{S}_n^+$ is a positive definite symmetric matrix and $r_a(x) = \sum_{\alpha \in I} a_\alpha x^\alpha$ is a polynomial, cf. (2). Moreover, it computes the corresponding Riemannian metric. The software uses the subgradient algorithm for geodesically convex functions on the manifold $\mathbb{R}^{|I|} \times \mathcal{S}_n^+$ to compute optimal p and coefficients a_α, in the sense that M delivers the best upper bound on the restoration entropy obtainable by a metric of this form.

4. Illustrative example

We show how to use the class library for the estimation of the restoration entropy for the three-dimensional, continuous-time Lorenz system. The dynamics are given by the equations

$$\dot{x}_1 = \sigma(x_2 - x_1)$$
$$\dot{x}_2 = x_1(\rho - x_3) - x_2$$
$$\dot{x}_3 = x_1 x_2 - \beta x_3$$

where σ, ρ and β are positive parameters. This example is included in the published code, but with some additional commands to store the results and keep track of the best estimate of the restoration entropy. In the context of networked or multi-agent systems, one can think about this system as one agent and its restoration entropy as the minimum data rate needed to communicate the state of this agent accurately. First, one defines the class `Lorenz` in *DynSys4EntSys.h*:

```
class Lorenz : public DynSys4EntSys {
public:
double sigma, rho, beta, K_Rad;
Lorenz(double _sigma = 10.0, double _rho = 28.0,
    double _beta = 8.0 / 3.0);
vec f(const vec &cx);
vec CoordTrans(const vec &ox);
mat Amat(const vec &cx);
};
```

Here the standard values $\sigma = 10$, $\rho = 28$, and $\beta = \frac{8}{3}$ are set as defaults. Then the methods of the class `Lorenz` are implemented in *DynSys4EntSys.cpp*:

```
Lorenz::Lorenz(double _sigma, double _rho, double
    _beta) : sigma(_sigma), rho(_rho), beta(_beta)
    , DynSys4EntSys(3) {
K_Rad = sqrt(beta / 2.0) * (sigma + rho);
}

vec Lorenz::f(const vec &cx) {
vec fx(dim);
fx(0) = sigma * (-cx(0) + cx(1));
fx(1) = rho * cx(0) - cx(1) - cx(0) * cx(2);
fx(2) = -beta * cx(2) + cx(0) * cx(1);
return fx;
}

mat Lorenz::Amat(const vec &cx) {
return { {-sigma, sigma, 0},
            {rho - cx(2), -1, -cx(0)},
            {cx(1), cx(0), -beta} };
}

vec Lorenz::CoordTrans(const vec &ox) {
  ox = ox   vec cx(dim);
  cx(0) = ox(0) * sin(ox(1)) * cos(ox(2));
  cx(1) = ox(0) * sin(ox(1)) * sin(ox(2));
  cx(2) = ox(0) * cos(ox(1)) + (sigma + rho);
  return cx;
}
```

In the constructor of `Lorenz`, one must initialize the base class `DynSys4EntSys` with the dimension of the system, stored in `dim`. The methods `Lorenz::f` and `Lorenz::Amat` are simply the right-hand-side of the differential equation $\dot{x} = f(x)$ defining the system and its Jacobian $Df(x)$, respectively.

The method `Lorenz::CoordTrans` is used to map the cube $[0, 1]^3$ to a forward-invariant set of the dynamics. From [9, Sec. II.2.2], it is known that the ball K with center $(0, 0, \sigma + \rho)$ and radius $K_{\text{Rad}} = (\sigma + \rho)\sqrt{\beta/2}$ is forward-invariant and `Lorenz::CoordTrans` maps the cube $[0, 1]^3$ onto this ball, see next step. Note that `%` is element-wise multiplication in Armadillo.

Next, we set up the optimization by making an instance of the class `Lorenz` and initialize an instance of the class `Esys` in `main` in *EstEntSGex.cpp*:

```
Lorenz sys;
umat r_a = MulInds(2, sys.dim);
uvec resol = {500, 50, 100};
EntSysCONT Esys(&sys, r_a, resol);
```

With the command `MulInds(2, sys.dim)`, we generate a matrix containing in its rows all multi-indices of dimension 3 (=`sys.dim`) of length ≤ 2, except for the zero multi-index that is not needed. This corresponds to the polynomial r_a in (2) with $I := \{\alpha \in \mathbb{N}_0^3 : 0 < |\alpha| \leq 2\}$. The (unsigned) integer vector `resol`, set as (500, 50, 100), specifies the distribution of the points in the cube $[0, 1]^3$; there are 500 points uniformly distributed on the first interval, 50 on the second and 100 on the third. Summed up, there are $500 \cdot 50 \cdot 100 = 2.5 \cdot 10^6$ points in the cube. In effect, this means that there are 500 points uniformly distributed on the interval $[0, K_{\text{Rad}}]$, 50 points on $[0, \pi]$, and 100 points on $[0, 2\pi]$, before the cuboid $[0, K_{\text{Rad}}] \times [0, \pi] \times [0, 2\pi]$ is mapped to the ball, cf. `Lorenz::CoordTrans` above. At each of these points, the function $g(x; a_j, p_j)$ will be evaluated in the search for the maximum. We then create an instance `Esys` of the class `EntSysCONT`; for a discrete-time system, we would similarly create an instance of the class `EntSysDISC`.

The iterations of the subgradient algorithm are now very simple to implement; note that we fix the number of iterations N:

```
int N = 4000;    // no. of iterations
double ta = 2.0, tb = 0.0; // t_j=ta/(j+tb)
int k0;
double MaxVal, EstEnt, t;
vec cx, s1;
mat s2;
double t;
for (int j = 0; j < N; j++){
    tie(MaxVal, cx) = Esys.FindMaximum();
    tie(s1, s2, k0, EstEnt) = Esys.riem_subg(
cx);
    Norms1s2(s1, s2);
    Esys.StepForward(t, s1, s2); // iteration
j+1
    cout << "iteration " << j << " : Entropy
estimate "
         << EstEnt << endl;
    t = ta / (j + 1 + tb);
}
```

In these computations, `Esys.FindMaximum()` computes the maximum `MaxVal` of $g(x; a_j, p_j)$ from (4) and `cx` is the position where it is attained. With the command `Esys.riem_subg(cx)` a Riemannian subgradient `s1,s2` is computed at the point `cx` and `k0` is the number of positive eigenvalues (number of singular values > 1 for discrete-time systems), and `EstEnt` is the restoration entropy estimate; for the formulas for the subgradient, see [2]. Finally, we normalize the subgradient with `Norms1s2(s1, s2)`, fix the next time step `t` for the subgradient algorithm, that then is performed with `Esys.StepForward(t, s1, s2)`.

5. How to use

- To define the dynamical system to be used, define a class derived from `DynSys4EntSys`. The constructor for this class must call the constructor of `DynSys4EntSys` with the dimension of the system as its argument and it must define the methods `vec f(const vec&)` and `mat Amat(const vec&)`, the right-hand side of the system $\dot{x} = f(x)$ or $x_{t+1} = f(x_t)$ and its Jacobian $A(x) := Df(x)$. Further, `vec CoordTrans(const vec&)` must also be defined such that it maps the cube $[0, 1]^n$ onto a forward-invariant set of the system.
- To set up the optimization problem for a continuous-time system $\dot{x} = f(x)$, one creates an instance of the class `EntSysCONT`. Similarly, for a discrete-time system $x_{t+1} = f(x_t)$, one creates an instance of the class `EntSysDISC`. The constructors of both classes take the same arguments: a pointer to a `DynSys4EntSys` defining the dynamical system, a set of multi-indices stored in the rows of a matrix of unsigned integers `umat r_a`, and a vector of unsigned integers `uvec resol`.
  The definition of the dynamical system was discussed above. As an example for `r_a`, if one wants to use the polynomial $r_a(x, y, z) = a_1 x + a_2 xy^2 z + a_3 x^2 z^3$ in the generic form of the metric (2) for a three-dimensional system, one would set `r_a ={{1,0,0},{1,2,1},{2,0,3}}` . If one wants to use a polynomial of the form $\sum_{0<|\alpha|\leq L} a_\alpha x^\alpha$, the function call `MakeMulInds(L,sys.dim)` can be used to generate the appropriate multi-indices. The resolution of the points distributed in the forward-invariant set is specified by an (unsigned) integer vector, e.g. `resol={10,20,30}` distributes 10, 20, and 30 points uniformly in the first, second, and third dimension in $[0, 1]^3$, respectively. The actual distribution is then determined by the method `CoordTrans` of the system, that maps these points into the forward-invariant set. The maximum of $g(x; a_j, p_j)$ from (4) or (3) is searched by evaluation at these points. The default is to refine the search by subsequently evaluating $g(x; a_j, p_j)$ at the points of a scaled down version of the grid close to the point where the maximum value was found using the coarser grid. To turn this refined search off, set the optional fourth argument of the constructor as `false`. The optional fifth argument can be used to fix the number of threads used in the maximization; if not set by the user, it will be set automatically.
- Compile and link using `make`.
- Run `./EntEstSGex`.

A few comments are in order. We use the normalization $n(s) = (s_1/|s_1|, s_2/|s_2|)$ as discussed in Section 2. To change this, just edit the method `EntSys::Norms1s2(vec &s1, mat &s2)`. By default, the algorithm starts with $a_0 = 0$ and $p_0 = I$. Use the functions `EntSys::Set_a(const vec &_a)` and/or `EntSys::SetP(const mat &_P)` to start with other values.

6. Impact

Our software computes Riemannian metrics for dynamical systems that give optimal upper bounds on the restoration entropy within the limitations of a given class of conformal Riemannian metrics; moreover, it computes the corresponding Riemannian metric. As discussed in Section 1, restoration entropy quantifies the minimum data rate, above which a remote observer can reconstruct the state of the dynamical systems without a degradation of the initial error over time. This is particularly important in networked control systems. In general, the computation of restoration entropy is a difficult task and our software delivers a useful tool to automatically compute upper bounds on the data rate needed to observe the system.

7. Conclusions

We have published a class library in C++ for the computation of optimal Riemannian metrics for dynamical systems using the subgradient method for geodesically convex functions, which

provides an upper bound on the restoration entropy. The code is easily applicable to any discrete- or continuous-time dynamical system and we give three concrete examples of how a dynamical system can be analyzed with our software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The software to the manuscript "ResEntSG: Restoration Entropy Estimation for Dynamical Systems via Riemannian Metric Optimization" can be accessed at https://github.com/shafstein/EntEstSG.

Acknowledgments

This work was funded by the German Research Foundation (DFG) through the grant ZA 873/4-1.

References

[1] Matveev AS, Pogromsky AY. Observation of nonlinear systems via finite capacity channels, part ii: restoration entropy and its estimates. Automatica 2019;103:189–99.
[2] Kawan C, Hafstein S, Giesl P. A subgradient algorithm for data-rate optimization in the remote state estimation problem. SIAM J Appl Dyn Syst 2021 [in press] Preprint on arXiv:2103.11131.
[3] Bhatia R. Positive definite matrices, vol. 24. Princeton University Press; 2009.
[4] Ferreira OP, Oliveira PR. Subgradient algorithm on Riemannian manifolds. J Optim Theory Appl 1998;97(1):93–104.
[5] Ferreira OP, Louzeiro MS, Prudente LF. Iteration-complexity of the subgradient method on Riemannian manifolds with lower bounded curvature. Optimization 2019;68(4):713–29.
[6] Bridson MR, Haeflinger A. Metric spaces of non-positive curvature, vol. 319. Springer Science & Business Media; 2013.
[7] Sanderson C, Curtin R. Armadillo: a template-based C++ library for linear algebra. J Open Source Softw 2016;1:26. http://dx.doi.org/10.21105/joss.00026.
[8] Sanderson C, Curtin R. A user-friendly hybrid sparse matrix class in C++. In: Davenport Kauers, Labahn Urban, editors. Mathematical software – ICMS 2018. 2018, p. 422–30. http://dx.doi.org/10.1007/978-3-319-96418-8_50.
[9] Boichenko VA, Leonov GA, Reitmann V. Dimension theory for ordinary differential equations. Teubner-Verlag; 2005.